Conjunctive Queries: Fast Evaluation

(Chapter 18 of DBT)

Complexity of Query Evaluation

**Theorem:** CQ-Evaluation is NP-complete, and in PTIME in data complexity

**Proof:**

**(NP-membership)** Guess-and-check:
- Consider a database $D$, a CQ $Q(x_1,...,x_k) :- \text{body}$, and a tuple $(a_1,...,a_k)$ of values
- Guess a substitution $h : \text{terms(body)} \rightarrow \text{terms}(D)$
- Verify that $h$ is a match of $Q$ in $D$, i.e., $h(\text{body}) \subseteq D$ and $(h(x_1),...,h(x_k)) = (a_1,...,a_k)$

**(NP-hardness)** Reduction from 3-colorability

**(in PTIME)** For every substitution $h : \text{terms(body)} \rightarrow \text{terms}(D)$, check if $h(\text{body}) \subseteq D$ and $(h(x_1),...,h(x_k)) = (a_1,...,a_k)$
Complexity of Query Evaluation

**Theorem:** CQ-Evaluation is NP-complete, and in PTIME in data complexity.

Evaluating a CQ $Q$ over a database $D$ takes time $|D|^{O(|Q|)}$. 
Minimizing Conjunctive Queries

Database theory has developed principled methods for optimizing CQs:

- Find an equivalent CQ with minimal number of atoms (the core)
- Provides a notion of “true” optimality

\[
\begin{align*}
Q(x) & : - R(x,y), R(x,b), R(a,b), R(u,c), R(u,v), S(a,c,d) \\
\{y \mapsto b\} & \\
Q(x) & : - R(x,b), R(a,b), R(u,c), R(u,v), S(a,c,d) \\
\{v \mapsto c\} & \\
Q(x) & : - R(x,b), R(a,b), R(u,c), \quad \quad S(a,c,d)
\end{align*}
\]
Minimizing Conjunctive Queries

• But, a minimal equivalent CQ might not be easier to evaluate - remains NP-hard

• “Good” classes of CQs for which query evaluation is tractable (*in combined complexity*):
  – Graph-based
  – Hypergraph-based
(Hyper)graph of Conjunctive Queries

\[ Q \ ::= \ R(x,y,z), \ R(z,u,v), \ R(v,w,x) \]
“Good” Classes of Conjunctive Queries

- **Graph-based**
  - CQs of bounded treewidth - their graph has bounded treewidth

- **Hypergraph-based:**
  - CQs of bounded hypertree width - their hypergraph has bounded hypertree width
  - Acyclic CQs - their hypergraph has hypertree width 1

measures how close a graph is to a tree

measures how close a hypergraph is to an acyclic one
Acyclic Hypergraphs

A join tree of a hypergraph $H = (V,E)$ is a labeled tree $T = (N,F,L)$, where $L : N \rightarrow E$ such that:

1. For each hyperedge $e \in E$ of $H$, there exists $n \in N$ such that $e = L(n)$

2. For each node $u \in V$ of $H$, the set $\{n \in N \mid u \in L(n)\}$ induces a connected subtree of $T$
A join tree of a hypergraph $H = (V,E)$ is a labeled tree $T = (N,F,L)$, where $L : N \rightarrow E$ such that:

1. For each hyperedge $e \in E$ of $H$, there exists $n \in N$ such that $e = L(n)$

2. For each node $u \in V$ of $H$, the set $\{n \in N \mid u \in L(n)\}$ induces a connected subtree of $T$

condition 2 is violated
Acyclic Hypergraphs

A join tree of a hypergraph $H = (V,E)$ is a labeled tree $T = (N,F,L)$, where $L : N \rightarrow E$ such that:

1. For each hyperedge $e \in E$ of $H$, there exists $n \in N$ such that $e = L(n)$

2. For each node $u \in V$ of $H$, the set $\{n \in N \mid u \in L(n)\}$ induces a connected subtree of $T$

**Definition:** A hypergraph is acyclic if it has a join tree

prime example of a cyclic hypergraph
Acyclic Hypergraphs

A join tree of a hypergraph $H = (V,E)$ is a labeled tree $T = (N,F,L)$, where $L : N \rightarrow E$ such that:

1. For each hyperedge $e \in E$ of $H$, there exists $n \in N$ such that $e = L(n)$

2. For each node $u \in V$ of $H$, the set $\{n \in N \mid u \in L(n)\}$ induces a connected subtree of $T$

**Definition:** A hypergraph is **acyclic** if it has a join tree

but this is acyclic
Relevant Algorithmic Tasks

ACYCLICITY

**Input**: a conjunctive query \( Q \)

**Question**: is \( Q \) acyclic? or is \( H(Q) \) acyclic?

\[ \{ Q \in \text{CQ} \mid H(Q) \text{ is acyclic} \} \]

ACQ-Evaluation

**Input**: a database \( D \), an acyclic conjunctive query \( Q \), and a tuple \((a_1, \ldots, a_k)\) of values

**Question**: \((a_1, \ldots, a_k) \in Q(D)\)?
Checking Acyclicity

Via the GYO-reduction (Graham, Yu and Ozsoyoglu)

1. Eliminate nodes occurring in at most one hyperedge

2. Eliminate hyperedges that are empty or contained in other hyperedges
Checking Acyclicity

Via the **GYO-reduction** (Graham, Yu and Ozsoyoglu)

1. Eliminate nodes occurring in at most one hyperedge

2. Eliminate hyperedges that are empty or contained in other hyperedges
Checking Acyclicity

Via the GYO-reduction (Graham, Yu and Ozsoyoglu)

1. Eliminate nodes occurring in at most one hyperedge

2. Eliminate hyperedges that are empty or contained in other hyperedges
Checking Acyclicity

Via the **GYO-reduction** (Graham, Yu and Ozsoyoglu)

1. Eliminate nodes occurring in at most one hyperedge
2. Eliminate hyperedges that are empty or contained in other hyperedges
Checking Acyclicity

Via the *GYO-reduction* (Graham, Yu and Ozsoyoglu)

1. Eliminate nodes occurring in at most one hyperedge
2. Eliminate hyperedges that are empty or contained in other hyperedges
Checking Acyclicity

Via the **GYO-reduction** (Graham, Yu and Ozsoyoglu)

1. Eliminate nodes occurring in at most one hyperedge
2. Eliminate hyperedges that are empty or contained in other hyperedges
Checking Acyclicity

Via the **GYO-reduction** (Graham, Yu and Ozsoyoglu)

1. Eliminate nodes occurring in at most one hyperedge

2. Eliminate hyperedges that are empty or contained in other hyperedges
Checking Acyclicity

Via the GYO-reduction (Graham, Yu and Ozsoyoglu)

1. Eliminate nodes occurring in at most one hyperedge

2. Eliminate hyperedges that are empty or contained in other hyperedges
Checking Acyclicity

Via the GYO-reduction (Graham, Yu and Ozsoyoglu)

1. Eliminate nodes occurring in at most one hyperedge

2. Eliminate hyperedges that are empty or contained in other hyperedges
Checking Acyclicity

Via the **GYO-reduction** (Graham, Yu and Ozsoyoglu)

1. Eliminate nodes occurring in at most one hyperedge
2. Eliminate hyperedges that are empty or contained in other hyperedges
Checking Acyclicity

Via the GYO-reduction (Graham, Yu and Ozsoyoglu)

1. Eliminate nodes occurring in at most one hyperedge

2. Eliminate hyperedges that are empty or contained in other hyperedges
Checking Acyclicity

Via the GYO-reduction (Graham, Yu and Ozsoyoglu)

1. Eliminate nodes occurring in at most one hyperedge
2. Eliminate hyperedges that are empty or contained in other hyperedges

**Theorem:** A hypergraph $H$ is acyclic iff $\text{GYO}(H) = \emptyset$

$\downarrow$

checking whether $H$ is acyclic is feasible in polynomial time, and if it is the case, a join tree can be found in polynomial time

$\downarrow$

**Theorem:** ACYCLICITY is in PTIME
Checking Acyclicity

**Theorem:** ACYCLICITY is in PTIME

**NOTE:** actually, we can check whether a CQ is acyclic in time $O(||Q||)$

linear time in the size $Q$
Evaluating Acyclic CQs

Theorem: ACQ-Evaluation is in PTIME

**NOTE:** actually, if $H(Q)$ is acyclic, then $Q$ can be evaluated in time $O(||D|| \cdot ||Q||)$

linear time in the size of $D$ and $Q$
Yannakaki’s Algorithm

Dynamic programming algorithm over the join tree

Given a database $D$, and an acyclic Boolean CQ $Q$

1. Compute the join tree $T$ of $H(Q)$
2. Assign to each node of $T$ the corresponding relation of $D$
3. Compute semi-joins in a bottom up traversal of $T$
4. Return YES if the resulting relation at the root of $T$ is non-empty; otherwise, return NO
Yannakaki’s Algorithm: Step 1

\[ Q := R_1(x_1, x_2, x_3), R_2(x_2, x_3), R_2(x_5, x_6), R_3(x_3), R_4(x_2, x_4, x_3) \]
Yannakaki’s Algorithm: Step 2
Yannakaki’s Algorithm: Step 3
Yannakaki’s Algorithm: Step 3
Yannakaki’s Algorithm: Step 3
Yannakaki’s Algorithm: Step 3
Yannakaki’s Algorithm: Step 4

YES
Recap

• “Good” classes of CQs for which query evaluation is tractable - conditions based on the graph or hypergraph of the CQ

• Acyclic CQs - their hypergraph is acyclic, can be checked in linear time

• Evaluating acyclic CQs is feasible in linear time (Yannakaki’s algorithm)